Name : Deep Pawar
CWID : A20545137
Sub : CS536 Assignment 6

Date ____
Page ____

Q. 1.

a) → **Precondition** : $n \geq 0$

Postcondition : $x = fac(n)$

where,

$$fac(n) = \begin{cases} 1, & \text{if } n = 0 \\ n * fac(n-1), & \text{if } n > 0 \end{cases}$$

b) → **Loop invariant** :

The loop invariant p is :

$$p \equiv x = fac(y) \wedge 0 \leq y \leq n$$

This invariant ensures that :

x holds the factorial of y at each step

y lies within the valid range of $[0, n]$

Loop condition :

The loop will terminate when $y = n$, meaning the loop should run as long as $y \neq n$. Hence, the loop condition is :

$$\boxed{B \equiv y \neq n}$$

c) → **Bound expression** :

A bound expression t must :

1. Decrease monotonically with each iteration of loop
2. Eventually reach 0, ensuring the loop terminates

Given, the loop starts with $y = 0$ & increments y by 1 in each iteration until $y = n$, the distance from the final value n decreases by 1 in every step, Thus:

$$\boxed{t = n - y}$$

Steps to show the working of Bound expression:

① Initially, $t = n - y = n - 0 = n$, which is non-negative

② In each iteration, y increments by 1, so t decreases by 1.

③ When $y = n$, $t = n - y = 0$, & the loop terminates

Q. 2.

→ There are many ways to write a program from which one possible full proof outline is as follows:

$\{ n \geq 0 \}$

$x := 1; \; \{ n \geq 0 \wedge x = 1 \} \; y := 0; \; \{ n \geq 0 \wedge x = 1 \wedge y = 0 \}$

$\{ \text{inv } p \equiv x = fac(y) \wedge 0 \leq y \leq n \} \; \{ \text{bd } n - y \}$

while $y \neq n$ do

$\{ x = fac(y) \wedge 0 \leq y \leq n \wedge y \neq n \wedge n - y = t_0 \}$

$\{ x * (y+1) = fac(y+1) \wedge 0 \leq (y+1) \leq n \wedge n - (y+1) < t_0 \} \; x := x * (y+1);$

$\{ x = fac(y+1) \wedge 0 \leq (y+1) \leq n \wedge n - (y+1) < t_0 \} \; y := y + 1$

$\{ x = fac(y) \wedge 0 \leq y \leq n \wedge n - y < t_0 \}$

od

$\{ x = fac(y) \wedge 0 \leq y \leq n \wedge y = n \}$

$\{ x = fac(n) \}$

Q. 3.

→ Here

assume that $0 \leq i < size(b), 0 \leq j < size(b),$
$0 \leq k < size(b)$

We need to create a full proof outline for the following minimal proof outline :

$$\{p\}\ b[i] := b[j];\ b[j] := b[k]\ \{b[i] > b[k]\}$$

$\therefore p_1 \equiv (b[i] > b[k])[b[k]/b[j]]$
$\equiv (\text{if } i = j \text{ then } b[k] \text{ else } b[i] \text{ fi}) >$
$(\text{if } k = j \text{ then } b[k] \text{ else } b[k] \text{ fi})$
$\leftrightarrow (\text{if } i = j \text{ then } b[k] \text{ else } b[i] \text{ fi}) > b[k]$

$\mapsto \text{if } i = j \text{ then } F \text{ else } b[i] > b[k] \text{ fi}$

$\mapsto i \neq j \wedge b[i] > b[k]$

Now, we can find optimized precondition p using backward assignment & by optimizing p_1 :

$\therefore p \equiv (i \neq j \wedge b[i] > b[k])(b[j]/b[i])$

$\equiv i \neq j \wedge b[j] > (\text{if } k = i \text{ then } b[j]$
$\text{else } b[k] \text{ fi})$

$\mapsto i \neq j \wedge (\text{if } k = i \text{ then } F \text{ else } b[j] > b[k] \text{ fi})$

$\mapsto i \neq j \wedge k \neq i \wedge b[j] > b[k]$

Q. 4.

→ $\{k < b[k] < b[j]\}\ \{P_1\}\ b[b[k]] := b[j]$

$\{b[k] \neq b[j]\}$

Here, we have to assume that,

$0 \leq j < size(b),\ 0 \leq k < size(b)$

Hence,

the full proof outline for the given minimal proof outline is as follows:

$$P_1 \equiv (b[k] \neq b[j])\,[b[j]\,/\,b[b[k]]]$$

$$\equiv (b[k])\,[b[j]\,/\,b[b[k]]] \neq (b[j])\,[b[j]\,/\,b[b[k]]]$$

$$\equiv (\text{if } k = b[k] \text{ then } b[j] \text{ else } b[k] \text{ fi}) \neq (\text{if } j = b[k] \text{ then } b[j] \text{ else } b[j] \text{ fi})$$

$$\rightarrow (\text{if } k = b[k] \text{ then } b[j] \text{ else } b[k] \text{ fi}) \neq b[j]$$

$$\rightarrow \text{if } k = b[k] \text{ then } b[j] \neq b[j] \text{ else } b[k] \neq b[j] \text{ fi}$$

$$\rightarrow \text{if } k = b[k] \text{ then } F \text{ else } b[k] \neq b[j] \text{ fi}$$

$$\rightarrow k \neq b[k] \wedge b[k] \neq b[j]$$

Therefore,

Full proof outline is as follows:

$\{ k < b[k] < b[j] \}$ $\{ k \neq b[k] \wedge b[k] \neq b[j]$

$b[b[k]] := b[j]$

$\{ b[k] \neq b[j] \}$

Q. 5.

→ Evaluation graph for $\langle S, \sigma \rangle$ where

$$S \equiv [x := 1 \,\|\, x := -1];\ y := y + x$$

$$
\begin{tikzcd}
& \langle [x := 1 \,\|\, x := -1];\ y := y + x,\ \sigma \rangle \arrow[dl] \arrow[dr] & \\
\langle [E \,\|\, x := -1];\ y := y + x;\ \sigma[x \mapsto 1] \rangle \arrow[d] & & \langle [x := 1 \,\|\, E];\ y := y + x,\ \sigma[x \mapsto -1] \rangle \arrow[d] \\
\langle [E \,\|\, E];\ y := y + x,\ \sigma[x \mapsto -1] \rangle \arrow[d] & & \langle [E \,\|\, E];\ y := y + x,\ \sigma[x \mapsto 1] \rangle \arrow[d] \\
\langle E, \sigma[x \mapsto -1][y \mapsto \sigma(y) - 1] \rangle & & \langle E, \sigma[x \mapsto 1][y \mapsto \sigma(y) + 1] \rangle
\end{tikzcd}
$$

Q. 6.

$\rightarrow$ Evaluation graph for $\langle W, \{x = 0, y = 1, n = 2\} \rangle$
where $W \equiv$ while $x < n$ do $[x := x + 1 \parallel y := y * 2]$ od

$\langle$while $x < n$ do $[x := x + 1 \parallel y := y * 2]$ od, $\{x = 0, y = 1, n = 2\} \rangle$

$\langle [x := x + 1 \parallel y := y * 2]; W, \{x = 0, y = 1, n = 2\} \rangle$

$\langle [E \parallel y := y * 2]; W, \{x = 1, y = 1, n = 2\} \rangle$

$\langle [x := x + 1 \parallel E]; W, \{x = 0, y = 2, n = 2\} \rangle$

$\langle W, \{x = 1, y = 2, n = 2\} \rangle$

$\langle [x := x + 1 \parallel y := y * 2]; W, \{x = 1, y = 2, n = 2\} \rangle$

$\langle [E \parallel y := y * 2]; W, \{x = 2, y = 2, n = 2\} \rangle$

$\langle [x := x + 1 \parallel E]; W, \{x = 1, y = 4, n = 2\} \rangle$

$\langle W, \{x = 2, y = 4, n = 2\} \rangle$

$\langle E, \{x = 2, y = 4, n = 2\} \rangle$

Q. 7.

a) Are these two threads disjoint ?

→ $\boxed{\text{Yes}}$

Explanation:

Here, the two threads are as follows:

$$S_1 \equiv \{x = 0\} \; y := x + 2 \; \{y = 2\}$$

$$S_2 \equiv \{x < 0\} \; z := 0 \; \{z > x\}$$

Two threads are disjoint if they operate on entirely separate variables & do not share any dependencies or modify each other's values.

Here,

S_1 modifies y, while S_2 modifies z. Thus, they modify distinct variables.

Neither thread depends on or alters the variable used by the other.

Date
Page

Therefore, the two threads are disjoint, as they do not share or interact through variables.

b) Do they have disjoint conditions

→ Yes

Explanation:

Two threads have disjoint conditions if their preconditions cannot both be true at the same time.

Here,

the precondition of S_1 is $x = 0$

the precondition of S_2 is $x < 0$

It is impossible for x to simultaneously satisfy $x = 0$ & $x < 0$. Therefore, the threads have disjoint conditions, as their preconditions cannot overlap.

Date

Page

Q. 8 Here, the threads written in proof outlines are :

$S_1^* \equiv \{p_1\}$ if B_1 then $\{p_2\}$ $\langle T_1 \rangle$ else $\{p_3\}$ skip fi $\{p_4\}$

$S_2^* \equiv \{q_1\}$ $\langle T_2 \rangle$; $\{$inv $q_2\}$ while B_2 do $\{q_3\}$ $\langle T_3 \rangle$ od $\{q_4\}$

a) list the interference between freedom checks to decide whether S_1^* interferes with S_2^*.

→

Interference Freedom checks for S_1^* interfering with S_2^* :

We need to determine if the atomic statement T_1 in thread S_1^* can interfere with the conditions, invariants, or postconditions in thread S_2^* :

The interference freedom checks are :

① $\{p2 \wedge q1\}$ T_1 $\{q1\}$:

This check ensures that executing T_1 in the "then" branch of S_1^* does not modify variables in a way that invalidates the precondition q1 of S_2^*.

② $\{p2 \wedge q2\}$ T_1 $\{q2\}$:

This ensures that executing T_1 does not invalidate the invariant q2 during the loop in S_2^*.

③ $\{p2 \wedge q3\}$ T_1 $\{q3\}$:

This checks make sure that T_1 does not invalidate the condition q3 used inside the loop body in S_2^*.

④ $\{p2 \wedge q4\}$ T_1 $\{q4\}$:

This ensures that T_1 does not invalidate the post-condition q4 in S_2^*.

Date
Page

These checks ensure that the atomic statement T_1 in S_1^* does not interfere with the preconditions, invariants or postconditions in thread S_2^*.

Skip interference check:

⑤ $\{p3 \wedge q\}$ skip $\{q\}$ for the same q:
Since skip does not perform any operation, this check is trivially satisfied. The invariants q_1, q_3 & S_2^* are unaffected.

b] List the interference freedom checks to decide whether S_2^* interferes with S_1^*.

→

Interference freedom checks for S_2^* interfering with S_1^*:
We need to consider the atomic statements T_2 & T_3 in thread S_2^* & determine if they interfere with the conditions or postconditions in thread S_1^*:
The interference freedom checks are:

[1] For T_2:

① $\{q1 \wedge p1\}$ $T2$ $\{p1\}$:
This ensures that executing $T2$ does not modify variables in a way that invalidates the precondition $p1$ of S_1^*.

② $\{q1 \wedge p2\}$ $T2$ $\{p2\}$:
This check ensures that $T2$ does not affect the condition $p2$, which is used after evaluating $B1$ in S_1^*.

③ $\{q1 \wedge p3\}$ $T2$ $\{p3\}$:
This ensures that $T2$ does not interfere with the assertion $p3$ in the "else" branch of S_1^*.

Date
Page

④ $\{q1 \wedge p4\}$ T2 $\{p4\}$:

This check ensures that T2 does not modify any variable that affects the postcondition p4.

[2] For T3 :

⑤ $\{q3 \wedge p1\}$ T3 $\{p1\}$:

This ensures that T3, which is executed in the loop body S_2^*, does not interfere with the precondition p1 of S_1^*.

⑥ $\{q3 \wedge p2\}$ T3 $\{p2\}$:

This check ensure that T3 does not modify variables in a way that affects the condition p2 in S_1^*.

⑦ $\{q3 \wedge p3\}$ T3 $\{p3\}$:

This ensures that T3 does not invalidate the assertion p3 in the "else" branch of S_1^*.

⑧ $\{q3 \wedge p4\}$ T3 $\{p4\}$:

This ensures that T3 does not interfere with the postcondition p4 in ~~S_1^*~~ S_1^*.

This checks ensure that the atomic statements T2 & T3 in S_2^* do not interfere with the assertion & conditions in S_1^*.